データを読み込む

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
#  以上3つのライブラリが「3種の神器」
```

In [2]:
```
Data=pd.read_csv("D:2022_数理統計学/StatData/StatData01_1.csv") # 読み出したいファイルのパス
```

In [3]:
```
Data # 読み込んだデータを見る。データには自動的に、0番から番号がつく。
```

Out[3]:

	新生児の体重(g)
0	3110
1	3100
2	3140
3	3050
4	2480
...	...
95	3170
96	2460
97	3360
98	3150
99	4180

100 rows × 1 columns

In [4]:
```
Data.head() # 読み込んだデータの先頭5行だけを表示
```

Out[4]:

	新生児の体重(g)
0	3110
1	3100
2	3140
3	3050
4	2480

In [5]:
```
# 日本語は避ける方がよい
Data = Data.rename(columns={'新生児の体重(g)':'Weight'}) # Data= とすることで上書きされる.
# Data.rename(columns={'新生児の体重(g)':'Weight'}, inplace=True) # これも同じ結果
Data.head()
```

Out[5]:

	Weight
0	3110
1	3100
2	3140
3	3050
4	2480

In [6]:
```
# ファイルを読み込む段階でカラム名を変更しておくことも可能（推奨）
Data=pd.read_csv("D:2022_数理統計学/StatData/StatData01_1.csv", # 読み出したいファイルのパス
    skiprows=1,              # データファイルの最初の1行を飛ばす
    names=['Weight'])        # カラム名を付ける
Data.head()
```

Out[6]:

	Weight
0	3110
1	3100
2	3140
3	3050
4	2480

基本的な統計量の計算

In [7]:
```
# 基本的な統計量 (0)  平均値
sum(Data['Weight'])/len(Data['Weight'])  # 総和を個数で割る
```

Out[7]: 3160.2

In [8]:
```
# 基本的な統計量 (1)  numpy
np.mean(Data['Weight']) # 平均値
```

Out[8]: 3160.2

In [9]:
```
# 基本的な統計量 (2)  numpy
print(np.mean(Data['Weight']),    # 平均値
      np.var(Data['Weight']),     # 分散  デフォルトで標本分散が出る
      np.std(Data['Weight']),     # 標準偏差
      np.min(Data['Weight']),     # 最小値
      np.max(Data['Weight']),     # 最大値
      np.median(Data['Weight']), # 中央値
      len(Data['Weight']),        # サイズ
     )
# 無駄に長い小数表示は、経過的には良いが最終的には適切に丸めること
```

```
3160.2 143911.96000000008 379.35729859856406 2270 4180 3160.0 100
```

In [10]:
```
# 基本的な統計量 (3) numpy DataFrame で出力
StatSummary=pd.DataFrame([
              ['平均値', np.mean(Data['Weight'])],
              ['分散', np.var(Data['Weight'])],
              ['標準偏差', np.std(Data['Weight'])],
              ['最小値', np.min(Data['Weight'])],
              ['最大値', np.max(Data['Weight'])],
              ['中央値', np.median(Data['Weight'])],
              ['サイズ', len(Data['Weight'])]
             ])
StatSummary
```

Out[10]:

	0	1
0	平均値	3160.200000
1	分散	143911.960000
2	標準偏差	379.357299
3	最小値	2270.000000
4	最大値	4180.000000
5	中央値	3160.000000
6	サイズ	100.000000

In [11]:
```
# 気に入らなければ整形しよう
StatSummary=StatSummary.rename(columns={0:'統計量'})
StatSummary=StatSummary.rename(columns={1:'Weight'})
StatSummary
```

Out[11]:

	統計量	Weight
0	平均値	3160.200000
1	分散	143911.960000
2	標準偏差	379.357299
3	最小値	2270.000000
4	最大値	4180.000000
5	中央値	3160.000000
6	サイズ	100.000000

In [12]:
```
# 気に入らなければ整形しよう
StatSummary=StatSummary.set_index('統計量') # カラム「統計量」をインデックスにする
StatSummary
```

Out[12]:

統計量	Weight
平均値	3160.200000
分散	143911.960000
標準偏差	379.357299
最小値	2270.000000
最大値	4180.000000
中央値	3160.000000
サイズ	100.000000

In [13]:
```
# 無駄に長い小数表示を修正（小数第3位を丸めて小数第2位までの表示）
StatSummary=pd.DataFrame([
              ['平均値', np.round(np.mean(Data['Weight']),2)],     # np.round(xxx,2) を使用
              ['分散', np.round(np.var(Data['Weight']),2)],
              ['標準偏差', np.round(np.std(Data['Weight']),2)],
              ['最小値', np.min(Data['Weight'])],
              ['最大値', np.max(Data['Weight'])],
              ['中央値', np.round(np.median(Data['Weight']),2)],
              ['サイズ', len(Data['Weight'])]
             ])
StatSummary=StatSummary.rename(columns={0:'統計量'})
StatSummary=StatSummary.rename(columns={1:'Weight'})
StatSummary=StatSummary.set_index('統計量') # カラム「統計量」をインデックスにする
StatSummary
```

Out[13]:

統計量	Weight
平均値	3160.20
分散	143911.96
標準偏差	379.36
最小値	2270.00
最大値	4180.00
中央値	3160.00
サイズ	100.00

In [14]:
```
# 基本的な統計量 (4)  pandas を用いてもよいが、分散の扱いに注意を要する（非推奨）
print(Data.mean(),          # 平均値
        Data.var(ddof=0),   # 分散 ddof=0 を指定しないと ddof=1 がデフォルトとなり不偏分散が出る
        Data.std(ddof=0),   # 標準偏差
        Data.min(),         # 最小値
        Data.max(),         # 最大値
        Data.median(),      # 中央値
        Data.count()        # サイズ
    )
```

```
Weight    3160.2
dtype: float64 Weight    143911.96
dtype: float64 Weight    379.357299
dtype: float64 Weight    2270
dtype: int64 Weight    4180
dtype: int64 Weight    3160.0
dtype: float64 Weight    100
dtype: int64
```

In [15]:
```
# 基本的な統計量 (5) pandas 8個の統計量を一括出力＝非推奨
# 標準偏差は不偏分散から計算しているので注意
Data.describe()
```

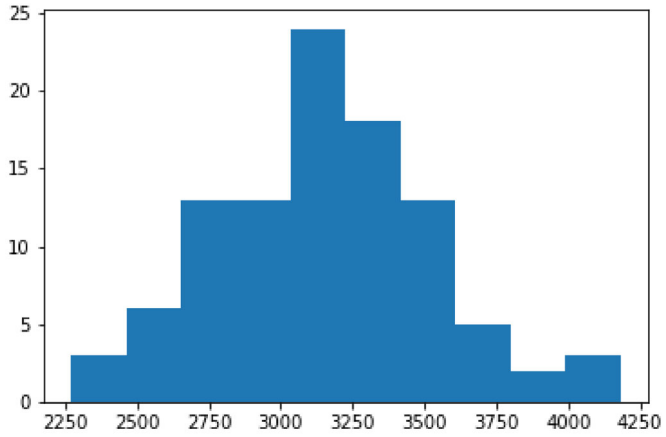
Out[15]:

	Weight
count	100.000000
mean	3160.200000
std	381.268431
min	2270.000000
25%	2897.500000
50%	3160.000000
75%	3367.500000
max	4180.000000

ヒストグラム

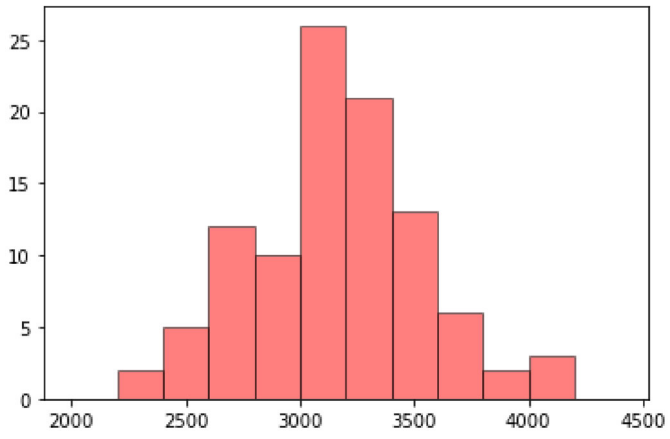
In [16]:
```
# ヒストグラム（自動生成＝非推奨）
plt.hist(Data['Weight'])  # 適当に階級を区切って自動的に表示してくれるが、まずいことが多い
# plt.show()  # 付帯情報を表示せず、描画のみを表示する
```

Out[16]:
```
(array([ 3.,  6., 13., 13., 24., 18., 13.,  5.,  2.,  3.]),
 array([2270., 2461., 2652., 2843., 3034., 3225., 3416., 3607., 3798.,
        3989., 4180.]),
 <BarContainer object of 10 artists>)
```

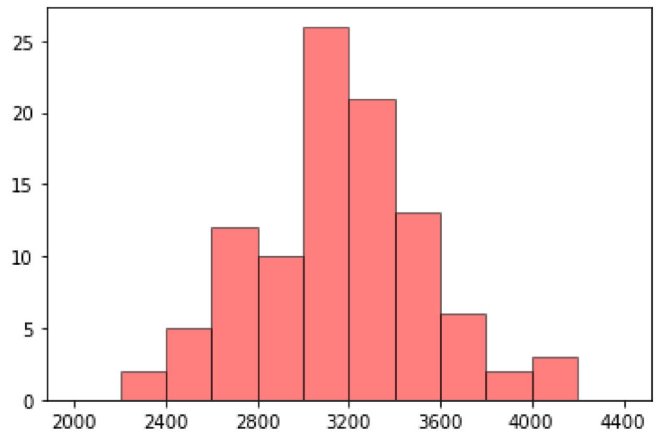


In [17]:
```
# ヒストグラム
# plt.figure(figsize=(8,6))    # 描画の大きさ。デフォルトは (6,4)
plt.hist(Data['Weight'],
         range=(2000,4400),    # 幅を指定
         bins=12,              # 階級数を指定
         color='red',          # 色を指定
         alpha=0.5,            # 色の濃さ (0～1)
         ec='k')               # 棒に枠線つける (k=black)
```

Out[17]:
```
(array([ 0.,  2.,  5., 12., 10., 26., 21., 13.,  6.,  2.,  3.,  0.]),
 array([2000., 2200., 2400., 2600., 2800., 3000., 3200., 3400., 3600.,
        3800., 4000., 4200., 4400.]),
 <BarContainer object of 12 artists>)
```

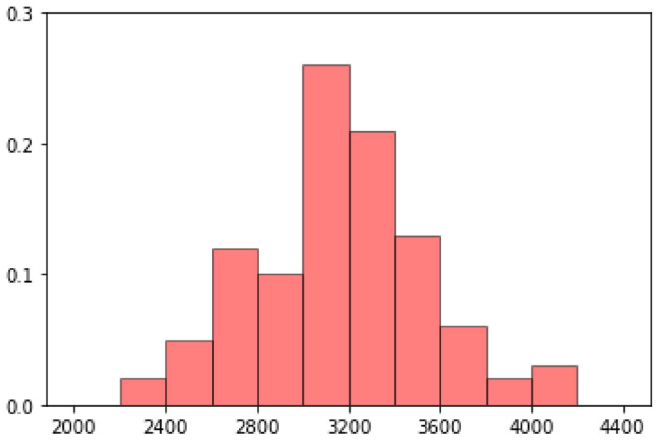


In [18]:
```
# 横軸や縦軸の目盛が不適正な場合は、手動で直す。
plt.hist(Data['Weight'],
     range=(2000,4400),
     bins=12,
     color='red',
     alpha=0.5,
     ec='k')
plt.xticks(np.arange(2000,4600,400))  # 横軸の調整（範囲と幅）
plt.yticks(np.arange(0,26,5))         # 縦軸の調整（範囲と幅）
plt.show()     # 付帯情報を表示せず、描画のみを表示する
```



In [19]:
```
#  相対度数のヒストグラム
RelW=np.ones_like(Data['Weight'])/len(Data['Weight']) # データ1個あたりウエイト（総和１）を均等に与
plt.hist(Data['Weight'],
             range=(2000,4400),
             bins=12,
             color='red',
             alpha=0.5,
             ec='k',
             weights=RelW)    # 度数のカウントをウエイトで行う
plt.xticks(np.arange(2000,4600,400))      # 横軸の調整（範囲と幅）
plt.yticks(np.arange(0,0.4,0.1))          # 縦軸の調整（範囲と幅）
```

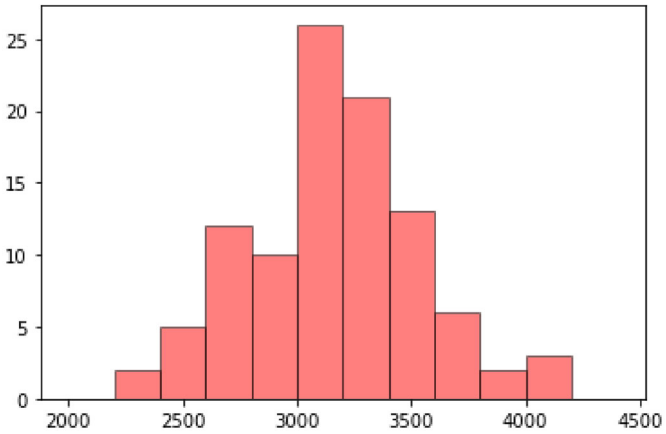
Out[19]:
```
([<matplotlib.axis.YTick at 0x2080175a640>,
  <matplotlib.axis.YTick at 0x2080175a220>,
  <matplotlib.axis.YTick at 0x20801756070>,
  <matplotlib.axis.YTick at 0x2080177f2e0>],
 [Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, '')])
```



度数分布表

In [20]:
```
# ヒストグラムから度数データを抽出
Y, X, _ = plt.hist(Data['Weight'],
             range=(2000,4400),
             bins=12,
             color='red',
             alpha=0.5,
             ec='k')
print(X,Y)
```

```
[2000.  2200.  2400.  2600.  2800.  3000.  3200.  3400.  3600.  3800.  4000.  4200.
 4400.] [ 0.  2.  5. 12. 10. 26. 21. 13.  6.  2.  3.  0.]
```



In [21]:
```
# 度数分布表（１）階級（ヒストグラムから読みだした X, Y を使う）
Jclass=[f'{X[i]}以上{X[i]+200}未満' for i in range(0,12)]  # f-string によって変数を文字列に埋め込
Jclass
```

Out[21]:
```
['2000.0以上2200.0未満',
 '2200.0以上2400.0未満',
 '2400.0以上2600.0未満',
 '2600.0以上2800.0未満',
 '2800.0以上3000.0未満',
 '3000.0以上3200.0未満',
 '3200.0以上3400.0未満',
 '3400.0以上3600.0未満',
 '3600.0以上3800.0未満',
 '3800.0以上4000.0未満',
 '4000.0以上4200.0未満',
 '4200.0以上4400.0未満']
```

In [22]:
```
# 度数分布表（２）階級値
Jclassmark=[X[i]+100 for i in range(0,12)]
Jclassmark
```

Out[22]:
```
[2100.0,
 2300.0,
 2500.0,
 2700.0,
 2900.0,
 3100.0,
 3300.0,
 3500.0,
 3700.0,
 3900.0,
 4100.0,
 4300.0]
```

In [23]:
```
FrequencyTable=pd.DataFrame({'階級値': Jclassmark, '度数':Y},
                            index=pd.Index(Jclass, name='階級'))
FrequencyTable
```

Out[23]:

階級	階級値	度数
2000.0以上2200.0未満	2100.0	0.0
2200.0以上2400.0未満	2300.0	2.0
2400.0以上2600.0未満	2500.0	5.0
2600.0以上2800.0未満	2700.0	12.0
2800.0以上3000.0未満	2900.0	10.0
3000.0以上3200.0未満	3100.0	26.0
3200.0以上3400.0未満	3300.0	21.0
3400.0以上3600.0未満	3500.0	13.0
3600.0以上3800.0未満	3700.0	6.0
3800.0以上4000.0未満	3900.0	2.0
4000.0以上4200.0未満	4100.0	3.0
4200.0以上4400.0未満	4300.0	0.0

In [24]:
```
# 度数が浮動小数点数はおかしいので、整数値に直す
YY=[int(Y[i]) for i in range(0,12)]  # 元の数列 Y から整数に直した新しい数列 YY を生成
YY
```

Out[24]: [0, 2, 5, 12, 10, 26, 21, 13, 6, 2, 3, 0]

In [25]:
```
# 階級の方も気になれば整数値に直す
XX=[int(X[i]) for i in range(0,12)]
XX
```

Out[25]: [2000, 2200, 2400, 2600, 2800, 3000, 3200, 3400, 3600, 3800, 4000, 4200]

In [26]:
```
# 度数分布表を再度作る
Jclass=[f'{XX[i]}以上{XX[i]+200}未満' for i in range(0,12)]  # f-string によって変数を文字列に埋め
Jclassmark=[XX[i]+100 for i in range(0,12)]
FrequencyTable=pd.DataFrame({'階級値': Jclassmark, '度数':YY},
                            index=pd.Index(Jclass, name='階級'))
FrequencyTable
```

Out[26]:

階級	階級値	度数
2000以上2200未満	2100	0
2200以上2400未満	2300	2
2400以上2600未満	2500	5
2600以上2800未満	2700	12
2800以上3000未満	2900	10
3000以上3200未満	3100	26
3200以上3400未満	3300	21
3400以上3600未満	3500	13
3600以上3800未満	3700	6
3800以上4000未満	3900	2
4000以上4200未満	4100	3
4200以上4400未満	4300	0

In [27]:
```
# 度数分布表の最後の階級は　〇〇以上〇〇以下をカウントしているので、表示は正しくない.
FrequencyTable.rename(index={'4200以上4400未満':'4200以上4400以下'},inplace=True)
FrequencyTable
```

Out[27]:

階級	階級値	度数
2000以上2200未満	2100	0
2200以上2400未満	2300	2
2400以上2600未満	2500	5
2600以上2800未満	2700	12
2800以上3000未満	2900	10
3000以上3200未満	3100	26
3200以上3400未満	3300	21
3400以上3600未満	3500	13
3600以上3800未満	3700	6
3800以上4000未満	3900	2
4000以上4200未満	4100	3
4200以上4400以下	4300	0

In [28]:
```
# 初めと終わりの度数 0 の階級は削除しよう
FrequencyTable.drop(['2000以上2200未満', '4200以上4400以下'], inplace=True)
FrequencyTable
```

Out[28]:

階級	階級値	度数
2200以上2400未満	2300	2
2400以上2600未満	2500	5
2600以上2800未満	2700	12
2800以上3000未満	2900	10
3000以上3200未満	3100	26
3200以上3400未満	3300	21
3400以上3600未満	3500	13
3600以上3800未満	3700	6
3800以上4000未満	3900	2
4000以上4200未満	4100	3

In [ ]: